字符串

2019年4月8日 15:57

-
- KMP算法

1. KMP模式匹配
   1) 为了方便判断边界条件，输入字串时**跳过下标0**
   2) 预处理next[i]=max{j}，j-1为前缀串的最后一个下标，且该前缀串与i-1为结尾的前缀串的后缀串能匹配，若不存在这种匹配，则令next[i]=0
   3) 引：设j0为next[i]的候选项，则next[j0]之前的都不是最大候选项
   4) 引：j是next[i]的候选项 iff j-1是next[i-1]的候选项（不越界时）
   5) 应用：next[i-1]及之前都打表成功后，next[i]的可能值是next[i-1]+1，next[next[i-1]]+1，next[next[next[i-1]]]+1……逐一判断即可
   6) 求next数组
      1) 初值：next[1]=0（此处的下标为字符串中字符的序号，没有0）
      2)
```
for(int i=2, j=0; i<=n; ++i){
        while(j>0 && a[i]!=a[j+1])
                j=next[j];
        if(a[i]==a[j+1])
                ++j;
        next[i]=j;}
```
   7) 求f数组（长串中各字符为结尾时能匹配到的最长原串前缀长度）
      1)
```
for(int i=1, j=0; i<=m; ++i){
        while(j>0 && (j==n || b[i]!=a[j+1]))
                j= next[j];
        if(b[i]==a[j+1])
                ++j;
        f[i]= j;
//      if(f[i]==n)     cout<<"找到模式串了";
}
```
2. 例：串串香匹配：判断t个s中能否找到w或反向的w，s中可能有?代表任意字符
   1) 将匹配从简单的相等改成相等或左边为问号
   2) 将w反过来，存在另一个数组，分别求next数组
   3)
```
const int MN = 100005;
char w[MN], rw[MN], s[MN];
int nxt[MN], rnxt[MN];  //正向和反向的next数组
int t;
inline bool pp(char c, char c2){
        return c=='?' || c==c2;
}        //重新定义匹配
```
   4)
```
scanf("%s%d",w+1,&t);
```

```
s[0]= w[0]= rw[0]= '!';    //空开下标0的字符
nxt[1]= 0;
int j= 0;
int len= strlen(w)-1;
for__(i,2,len){
        while(j>0 && w[i]!=w[j+1])
                j= nxt[i];
        if(w[i]==w[j+1])
                ++j;
        nxt[i]= j;}
for__(i,1,len)
        rw[i]= w[len+1-i];
rnxt[1]= 0;
j= 0;
for__(i,2,len){
        while(j>0 && rw[i]!=rw[j+1])
                j= rnxt[i];
        if(rw[i]==rw[j+1])
                ++j;
        rnxt[i]= j;}
while(t--){
        scanf("%s",s+1);
        int lens= strlen(s)-1;
        j= 0;
        bool ys= 0;
        for__(i,1,lens){
                while(j>0 && !pp(s[i], w[j+1]))
                        j= nxt[j];
                if(pp(s[i], w[j+1]))
                        ++j;
                if(j==len){
                        ys= 1;
                        break;}}
        if(ys){
                printf("YES\n");
                continue;}
        j= 0;
        for__(i,1,lens){
                while(j>0 && !pp(s[i], rw[j+1]))
                        j= rnxt[j];
                if(pp(s[i], rw[j+1]))
                        ++j;
                if(j==len){
                        ys= 1;
                        break;}}
        if(ys)
                printf("YES\n");
        else
                printf("NO\n");}
```

-
- 常用函数

3. 求子串

```
std::string::substr
string substr (size_t pos = 0, size_t len = npos) const;
```

Generate substring

1) Returns a newly constructed string object with its value initialized to a copy of a substring of this object.
2) The substring is the portion of the object that starts at character position pos and spans len characters (or until the end of the string, whichever comes first).
3) Parameters
    1) pos
        1) Position of the first character to be copied as a substring.
        2) If this is equal to the string length, the function returns an empty string.
        3) If this is greater than the string length, it throws out_of_range.
        4) Note: The first character is denoted by a value of 0 (not 1).
    2) len
        1) Number of characters to include in the substring (if the string is shorter, as many characters as possible are used).
        2) A value of string::npos indicates all characters until the end of the string.
4) size_t is an unsigned integral type (the same as member type string::size_type).
   来自 <http://www.cplusplus.com/reference/string/string/substr/>
5) 顺带一提java除了substr外还有subString(l,r)自动给负数下标转换成0，l>r时自动交换lr，再返回[l,r)

4. 找子串

   std::string::find

- C++11

```
     string (1)  size_t find (const string& str, size_t pos = 0) const noexcept;
   c-string (2)  size_t find (const char* s, size_t pos = 0) const;
     buffer (3)  size_t find (const char* s, size_t pos, size_type n) const;
  character (4)  size_t find (char c, size_t pos = 0) const noexcept;
```

1) Find content in string
2) Searches the string for the first occurrence of the sequence specified by its arguments.
3) When pos is specified, the search only includes characters at or after position pos, ignoring any possible occurrences that include characters before pos.
4) Notice that unlike member find_first_of, whenever more than one character is being searched for, it is not enough that just one of these characters match, but the entire sequence must match.
5) Parameters
    1) str
        1) Another string with the subject to search for.
    2) pos
        1) Position of the first character in the string to be considered in the search.
        2) If this is greater than the string length, the function never finds

matches.
            3) Note: The first character is denoted by a value of 0 (not 1): A value of 0 means that the entire string is searched.
        3) s
            1) Pointer to an array of characters.
            2) If argument n is specified (3), the sequence to match are the first n characters in the array.
            3) Otherwise (2), a null-terminated sequence is expected: the length of the sequence to match is determined by the first occurrence of a null character.
        4) n
            1) Length of sequence of characters to match.
        5) c
            1) Individual character to be searched for.
    6) Return Value
        1) The position of the first character of the first match.
        2) If no matches were found, the function returns string::npos.
        3) size_t is an unsigned integral type (the same as member type string::size_type).

来自 <http://www.cplusplus.com/reference/string/string/find/>

◆

◆ 例题

1. 求长度n的随机字符串中，长度不小于m的不同子串数量
    1) 因为是随机的，所以长度大于8的其实没有重复子串……
    2) 本来想用(t=s.find(sb,t+1))!=s.npos来遍历每个相同字符串的，超时
    3) 果然还是set大法好
    4)
```
ull ans=0;
for__(l,m,min(n,10)){
        for__(i,0,n-l)
                ss.insert(s.substr(i,l));
        ans+=ss.size();
        ss.clear();}
for__(l,max(m,11),n)
        ans+=n-l+1;
cout<<ans;
```

i.
ii.
iii.
iv.
v.
vi.
vii.
viii.
ix.

x. ---------我是底线----------